ENPM665 0101 – Team 14

Final Group Project:
Cloud Security Hardening Plan

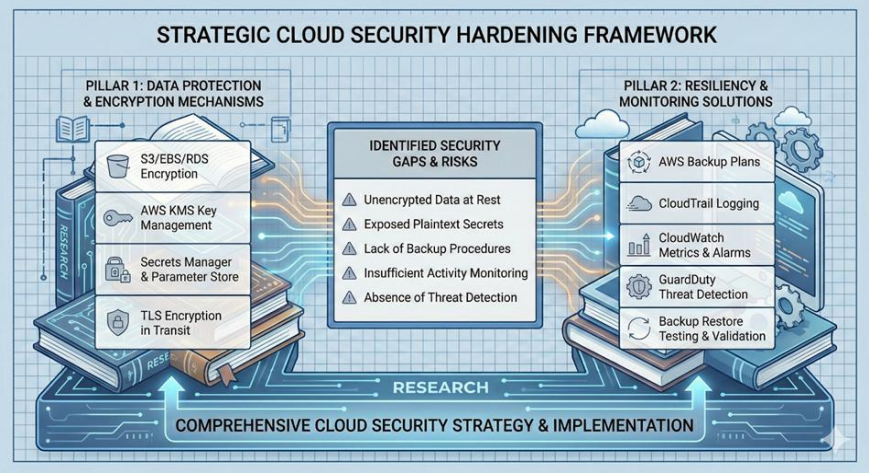
Name	Directory ID	UID
Viraj Sanjay Kurhade	vkurhade	122010192
Kalpesh Bharat Parmar	kalpesh	122046788
Chandana Charitha Peddinti	charitha	121288875
Ashley Nichole Rose	anrose	121381811
Vineet Agarwal	vineet54	121386052
Fahad Shaker	fshaker	119529003

Table of Content

List of Figures

1. Executive Summary & Focus Area Selection

The midterm assessment showed clear patterns across the environment: sensitive data was stored unencrypted, secrets were exposed in plaintext, activity across the infrastructure was largely invisible due to missing logs, and the system had no backup or recovery plan. These issues directly threatened confidentiality, integrity, availability, and the organization's ability to recover from operational incidents. To address the highest-impact gaps, this hardening plan concentrates on two well-defined pillars:



To address the highest-impact risks, this project centers on two focus areas:

Focus Area 1: Data Protection & Encryption

The midterm revealed unencrypted EBS volumes, unprotected S3 objects, plaintext secrets in configuration files, disabled TLS, and a complete absence of key management. This focus area aims to safeguard sensitive data at rest and in transit by enforcing encryption for storage services (S3, RDS, EBS), implementing TLS, centralizing encryption keys with AWS KMS, and shifting all secrets to AWS Secrets Manager or SSM Parameter Store. These measures close the gaps that previously enabled data leakage, credential exposure, and unauthorized access.

3 Key Improvements were selected in the focus area of Data Protection & Encryption

- Enable Encryption for S3, RDS, and EBS
- Use AWS KMS for Key Management
- Secure Sensitive Data with Secrets Manager or Parameter Store

Focus Area 2: Disaster Recovery & Monitoring

The environment operates without backups, snapshots, monitoring, or threat detection. A single EC2 instance, no automated recovery points, and no audit trails meant that a failure or compromise could result in permanent data loss with no ability to investigate. This focus area introduces AWS Backup, CloudTrail, CloudWatch, and GuardDuty, along with recurring restore simulations. Together, these improvements build resilience, provide visibility into activity, and ensure the system can recover quickly if an outage or attack occurs.

3 Key Improvements were selected in the focus area of Disaster Recovery & Monitoring

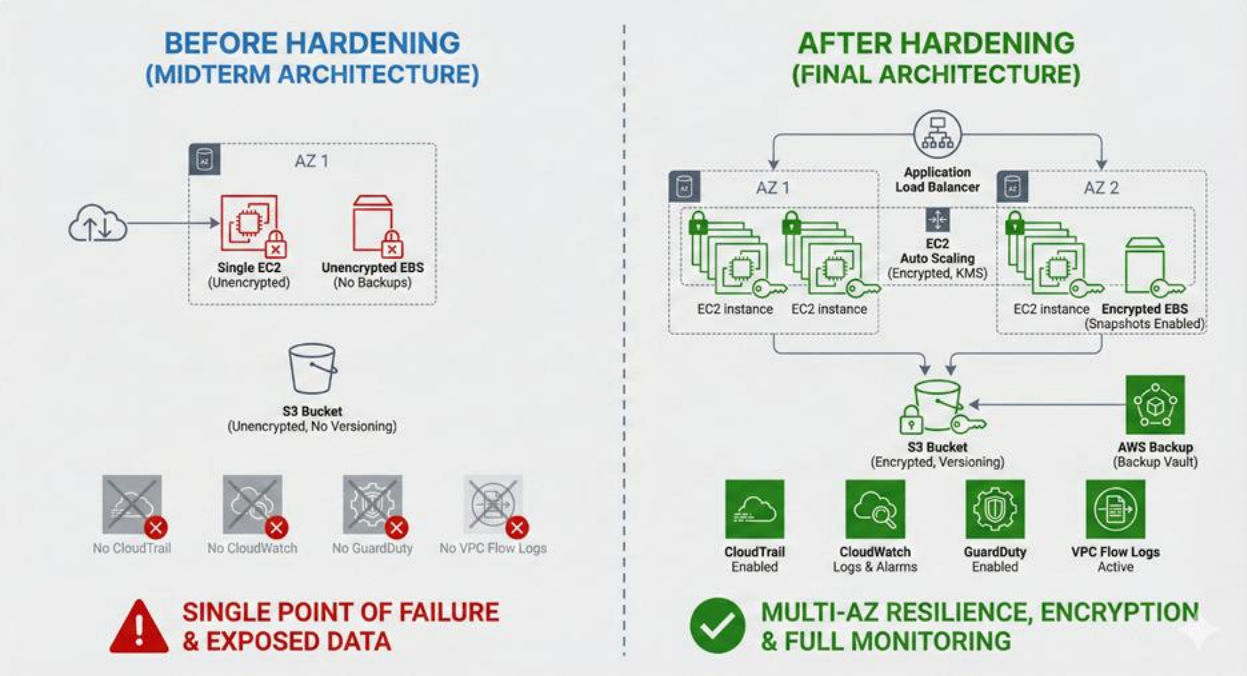
- Configure AWS Backup and Recovery Plans
- Enable CloudTrail and CloudWatch Monitoring
- Deploy GuardDuty for Threat Detection

Justification for Selecting These Two Focus Areas

We chose **Data Protection & Encryption** and **Disaster Recovery & Monitoring** because they addressed the most critical weaknesses from the midterm. The environment stored sensitive data without encryption, exposed credentials in plaintext, and lacked any secure method for managing secrets — making data protection an immediate priority. At the same time, there were no backups, no logging, no threat detection, and no monitoring, meaning incidents would go unnoticed and unrecoverable. Together, these two areas delivered the largest, most meaningful improvement to confidentiality, integrity, and availability, and provided a clear, actionable path to strengthening the entire cloud environment.

2. Objectives & Key Improvements

The hardening plan is organized under two objectives. Each objective includes targeted improvements tied directly to the midterm findings.



Objective 1: Strengthen Data Protection Through Encryption & Secure Secret Handling

Key Improvement 1: Enable Encryption for S3, RDS, and EBS

Why this matters:
Midterm results showed unencrypted EBS volumes and S3 buckets, leaving storage vulnerable if copied, accessed, or modified. Enforcing encryption protects data even if compromised.

Resource	Midterm State	Improvement
S3	Buckets not enforced with encryption	Enable SSE-S3 or SSE-KMS
EBS	EC2 root volume unencrypted	Use EBS encryption with KMS
RDS (when DB is migrated)	No at-rest database encryption	Enable RDS encryption

Key Improvement 2: Use AWS KMS for Key Management

Actions:

- Create or use an AWS-managed KMS key
- Use the CMK to encrypt:
    - S3 buckets
    - EBS volumes
    - RDS database
    - AWS Backup vaults

Why this matters:
KMS adds lifecycle-managed, auditable keys, reducing accidental exposure and unauthorized decryption.

Key Improvement 3: Secure Sensitive Data with Secrets Manager or Parameter Store

Actions:

- Move plaintext passwords, DB credentials, API tokens, and environment variables into Secrets Manager/SSM
- Restrict access using IAM least-privilege
- Enable automatic rotation where applicable

Why this matters:
This removes all plaintext secrets from EC2, configuration files, and user-data scripts, eliminating a major midterm risk.

Objective 2: Ensure Resilience with Disaster Recovery & Monitoring

Key Improvement 4: Configure AWS Backup and Recovery Plans

Component	Backup Type	Frequency	Retention
EBS volumes	Snapshots	Daily	30 days
RDS (after migration)	Automated backups	Daily + PITR	7–14 days
S3 Buckets	Versioning + Cross-Region Replication (optional)	Continuous	Based on policy

Why this matters:
These backups ensure reliable recovery points and eliminate the single-point-of-failure noted in the midterm.

Simulate Backup Restores

Actions:

- Test EC2 restoration from snapshot
- Test EBS volume restoration
- Verify RDS restoration when applicable
- Validate S3 version restore process

Why this matters:
A DR plan only works if validated. Restore tests confirm that recovery expectations (RTO/RPO) can be met.

Key Improvement 5: Enable CloudTrail and CloudWatch Monitoring

Actions:

- CloudTrail (multi-region) for all API activity
- CloudWatch metrics + alarms (CPU, status checks, backup failures)
- Push EC2 OS logs to CloudWatch

Why this matters:
Monitoring transforms a previously "blind" environment into one where anomalous activity is visible and actionable.

Key Improvement 6: Deploy GuardDuty for Threat Detection

GuardDuty detects:

- IAM anomalies
- Reconnaissance activity
- Port scans
- Malicious IP contact
- Credential compromise indicators

Why this matters: The midterm highlighted a complete absence of threat detection. GuardDuty closes this gap without manual rules.

3. Risk Alignment with Midterm Findings

This section maps midterm findings directly to the chosen focus areas and their associated improvements. This traceability ensures the final plan addresses the highest-impact risks noted in the assessment.

Midterm Finding	Risk Category	Impact	Final Improvement	Focus Area
Unencrypted EBS volume	Data exposure	Critical	Enable EBS encryption	Data Protection & Encryption
S3 objects unprotected	Data leakage	High	Enable SSE-KMS	Data Protection & Encryption
DB/data unencrypted	Confidentiality breach	High	RDS encryption	Data Protection & Encryption
Secrets stored in plaintext	Credential compromise	High	Store in Secrets Manager	Data Protection & Encryption
No CloudTrail	No audit trail	High	Enable CloudTrail	Disaster Recovery & Monitoring
No CloudWatch monitoring	No visibility	High	Enable CloudWatch	Disaster Recovery & Monitoring
No GuardDuty	No threat detection	High	Enable GuardDuty	Disaster Recovery & Monitoring
No backups or snapshots	Data loss possible	Critical	Configure AWS Backup	Disaster Recovery & Monitoring
No recovery testing	RTO/RPO unknown	High	Simulate restores	Disaster Recovery & Monitoring

4. Key Improvement

4.1 Data Protection - S3 and EBS Encryption

Implementation Plan

Objective: Ensure all new EBS volumes and S3 objects are automatically encrypted at rest using AWS KMS, preventing unauthorized access to data even if storage media is compromised.

Midterm Risk Addressed: Unencrypted EBS volumes storing database files and unprotected S3 objects are vulnerable to data leakage.

AWS Services Used: Amazon EC2, Amazon EBS, Amazon S3, AWS KMS

Implementation Steps

1. Enable Account-Level Default EBS Encryption

The existing EBS volume (vol-0119ab48aae112060) attached to VulnerableInstance was verified as unencrypted. Account-level default encryption was enabled to protect all future volumes.
Console Steps:

- Navigated to EC2 Console → Settings (under Account attributes)
- Located the "EBS encryption" section and clicked "Manage"
- Checked the box for "Always encrypt new EBS volumes"
- Selected the default AWS-managed KMS key (arn:aws:kms:us-east-1:466814371984:key/4466f459-8b38-41af-b4a8-af9d926bd9ad)
- Clicked "Update EBS encryption"

2. Create and Encrypt S3 Bucket

A new S3 bucket was created with default encryption enabled to protect all stored objects.
Console Steps:

- Created bucket: enpm665-group14-juiceshop-8827 in us-east-1
- Enabled Block Public Access (all settings)
- Navigated to Properties → Default encryption → Edit
- Enabled Server-side encryption with SSE-S3
- Enabled Bucket Key for cost optimization
- Saved changes

3. Apply Bucket Policy to Deny Unencrypted Uploads

To enforce encryption at the policy level, a bucket policy was applied that denies any upload requests without server-side encryption headers. Refer Appendix 1 for the bucket policy details.
Console Steps:

- Navigated to bucket's Permissions tab → Bucket policy → Edit
- Applied DenyUnencryptedUploads policy
- Confirmed with green banner: "Successfully edited bucket policy"

Validation and Testing

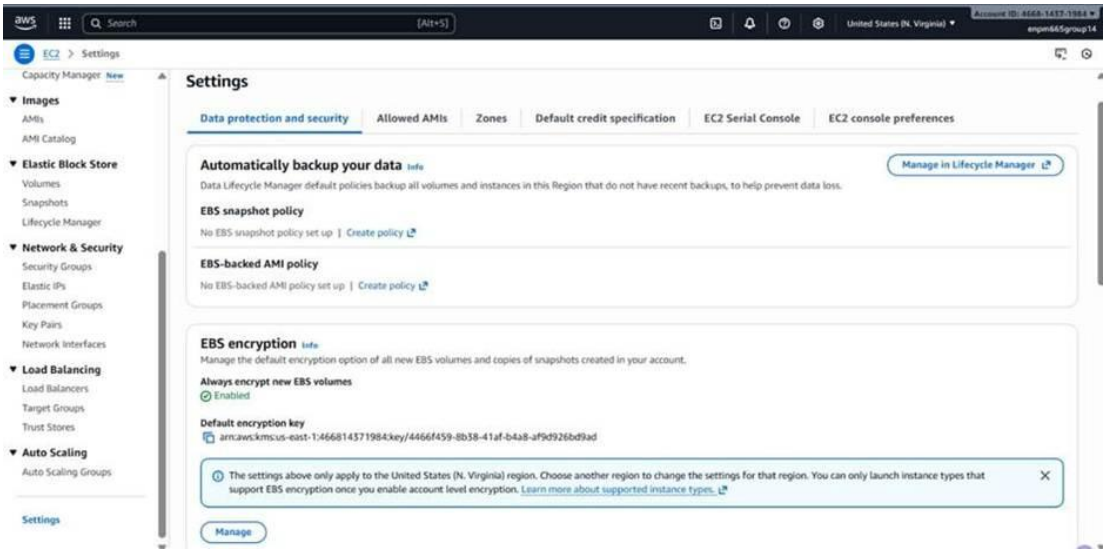


Figure 1: EC2 Settings showing EBS Default Encryption Enabled

```
PS C:\Windows\system32> aws ec2 get-ebs-encryption-by-default --region us-east-1
{
    "EbsEncryptionByDefault": true
}
```

Figure 2: CLI output confirming EbsEncryptionByDefault: true

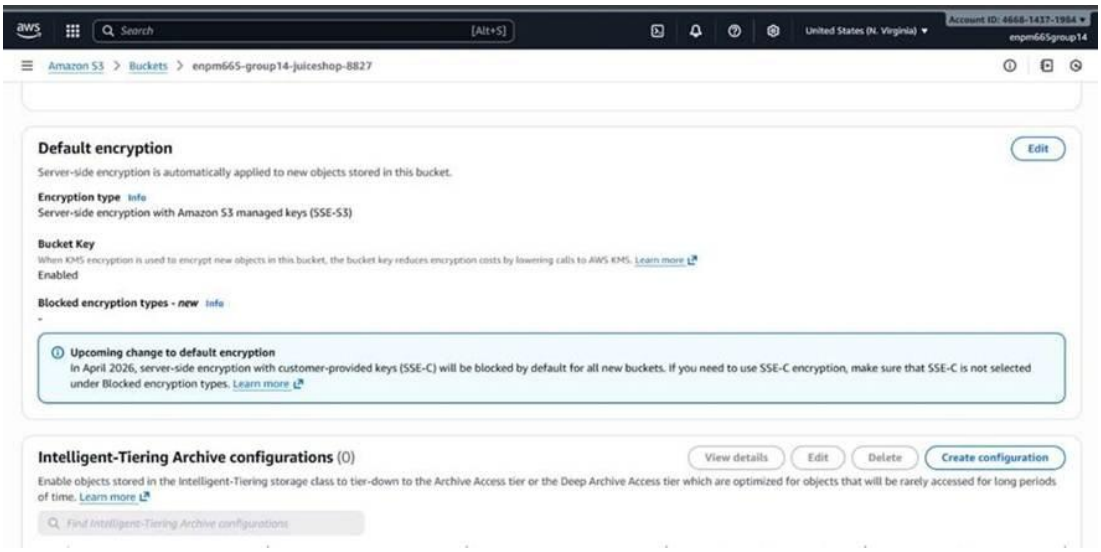


Figure 3: S3 Bucket Properties showing SSE-S3 Encryption Enabled

```
PS C:\Windows\system32> aws s3api get-bucket-encryption --bucket enpm665-group14-juiceshop-8827
{
    "ServerSideEncryptionConfiguration": {
        "Rules": [
            {
                "ApplyServerSideEncryptionByDefault": {
                    "SSEAlgorithm": "AES256"
                },
                "BucketKeyEnabled": true
            }
        ]
    }
}
```

Figure 4: CLI output confirming S3 encryption with AES256

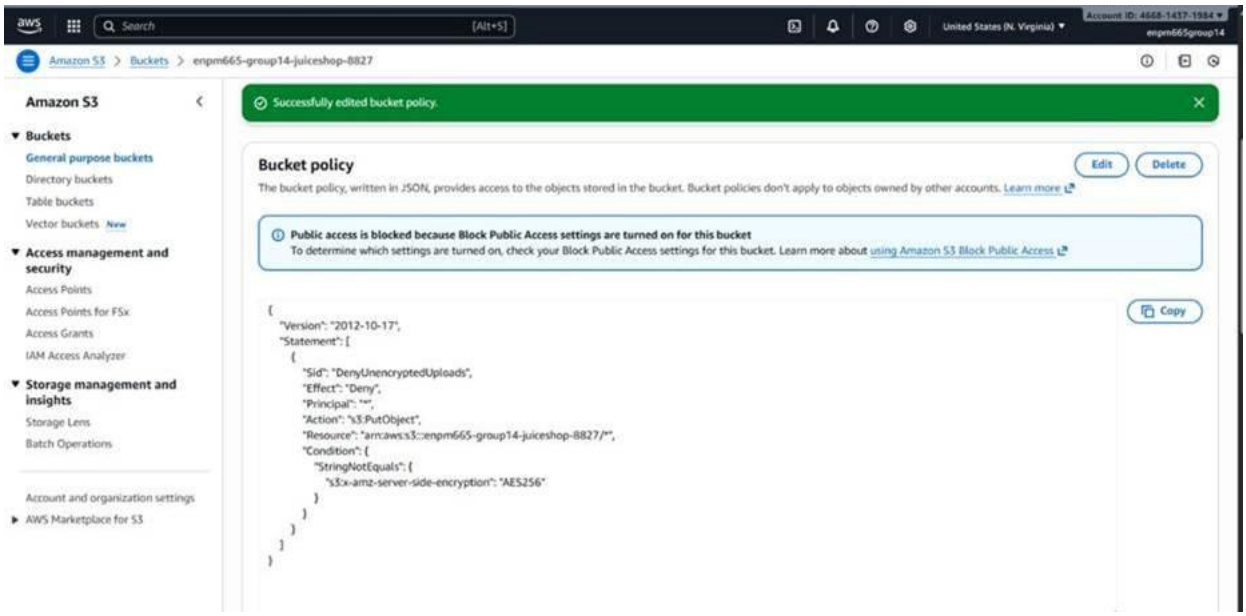


Figure 5: S3 Bucket Policy showing DenyUnencryptedUploads

Figure 6: CLI output confirming bucket policy

Impact Assessment

These changes had a meaningful impact on how securely our data is handled. By enabling default EBS encryption, we made sure that every new volume is protected automatically—removing the earlier risk of storing database files on unencrypted disks. Setting up the new S3 bucket with default encryption, along with a policy that blocks any unencrypted uploads, gave us consistent protection for all objects without relying on manual checks.

Overall, the environment is now much safer against accidental misconfigurations or data leakage. These improvements are simple to maintain, work automatically in the background, and strengthen our security posture without affecting performance. It's a solid step toward making the storage layer more resilient and aligned with best practices.

Summary

Summary

EBS default encryption and S3 server-side encryption were successfully enabled to address the critical data protection gaps identified in the midterm assessment. All future EBS volumes will be automatically encrypted using the AWS-managed KMS key, and the S3 bucket is now protected with SSE-S3 encryption plus a bucket policy that explicitly denies unencrypted uploads. These controls ensure data confidentiality at rest and provide defense-in-depth against misconfiguration.

Recommendations:

- Migrate the existing unencrypted EBS volume (vol-0119ab48aae112060) by creating an encrypted snapshot and launching a new encrypted volume to eliminate the residual risk.
- Consider upgrading from SSE-S3 to SSE-KMS with a customer-managed key for enhanced control over key rotation and access policies.
- Enable S3 versioning and cross-region replication to protect against accidental deletion and regional outages.
- Periodically audit encryption settings using AWS Config rules (ec2-ebs-encryption-by-default, s3-bucket-server-side-encryption-enabled) to ensure compliance drift is detected.

4.2 Data Protection – KMS Encryption of Sensitive Data

Objective:

Implement Customer Managed KMS Keys to ensure sensitive data is encrypted with keys fully controlled by JuiceShop, meeting organizational and compliance requirements.

Midterm Risk Addressed:

Lack of controlled, auditable encryption keys increases the risk of unauthorized data access and non-compliance with security standards.

AWS Services Used:

AWS KMS, Amazon RDS (Aurora).

Implementation Plan

1. Create a customer managed KMS key(s) with multi-region availability, and step through the configuration wizard.

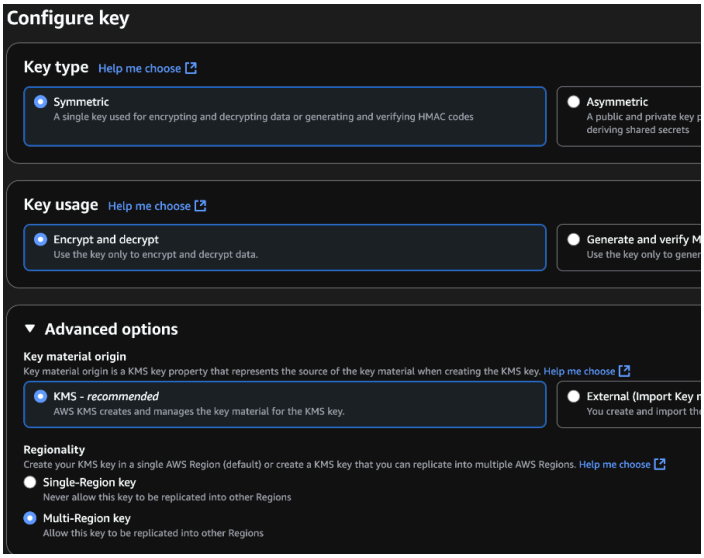


Figure 7: Configuring KMS key

2. When defining key administrators and permissions, ensure that only the most trustworthy database administrator users are selected (enforce employee background checks).

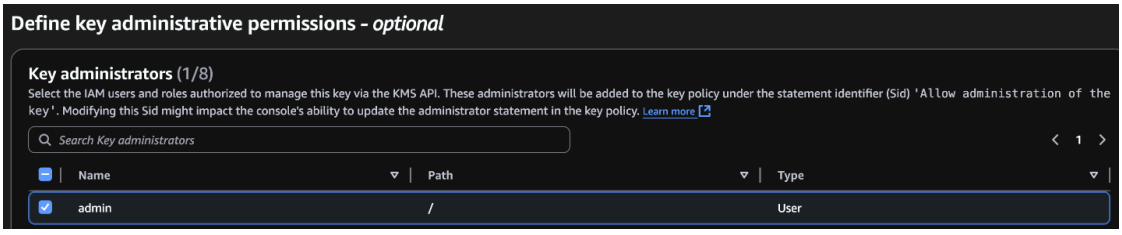


Figure 8: Setting up key administrators

3. Enable key rotation at least once every 12 months (perhaps more depending on regulatory legislation – consult with lawyers to ensure compliance)

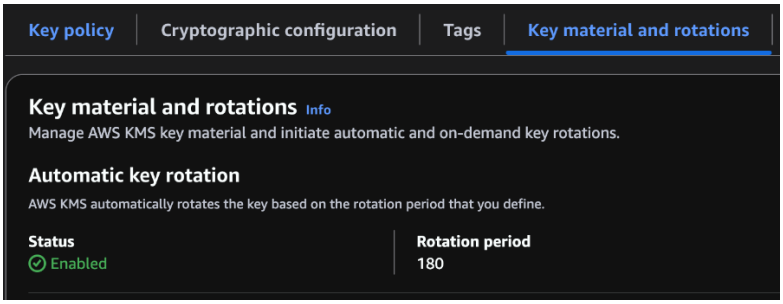


Figure 9: Automatic key rotation enabled

4. KMS key to encrypt the content

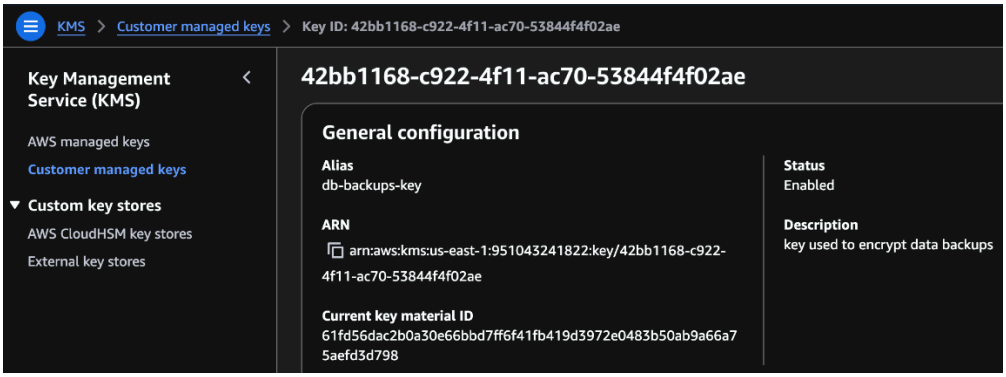


Figure 10: KMS Key to encrypt the content

5. When provisioning storage solutions with sensitive data (such as when migrating data from the local mariadb instance in EC2 to an Aurora RDS instance, moving the FTP files to a private s3 bucket, etc), be sure to select the appropriate KMS key to encrypt the content.

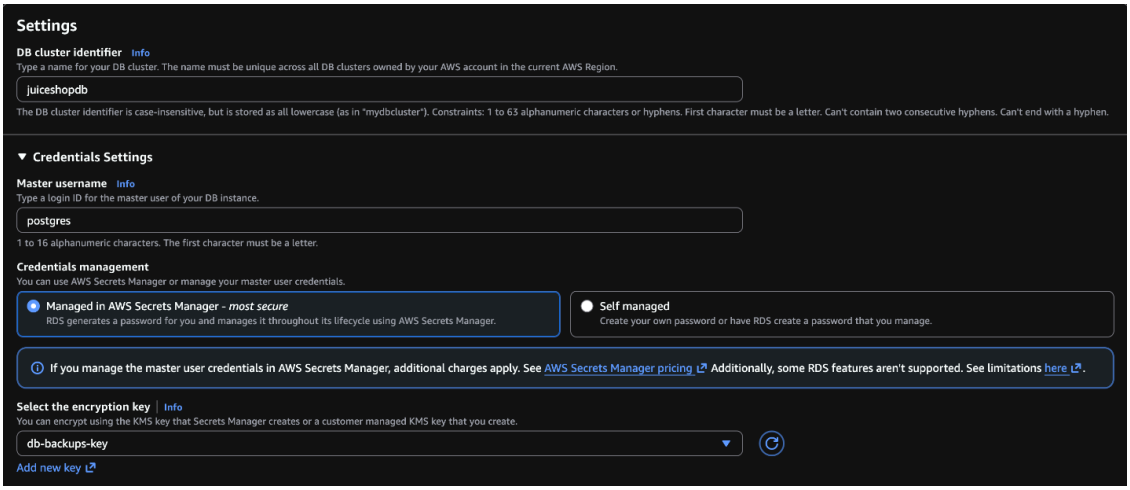


Figure 11: KMS key selection

Impact Assessment

Implementing Customer Managed KMS Keys made a clear and practical improvement to how JuiceShop protects sensitive data. By creating and managing our own encryption keys, we moved beyond the default AWS-managed setup and brought the process fully under our control. Giving key access only to a few trusted administrators reduced any insider risk, and enabling regular key rotation added long-term security without extra effort. Using these keys across RDS, S3, and other storage services ensured that everything stayed consistently encrypted and reduced the chances of any accidental mistakes. With CloudTrail recording all key activity, we now have better visibility and audit readiness as well. Overall, this change strengthened data confidentiality, aligned well with compliance needs, and gave us a more secure and reliable environment.

Cost considerations

The main cost component is separate DB instance for Amazon Aurora, the tentative cost is $550 per month for such a small deployment. This cost put a significat roadblock in securing all the data, but at least critical data should be protected in this manner.

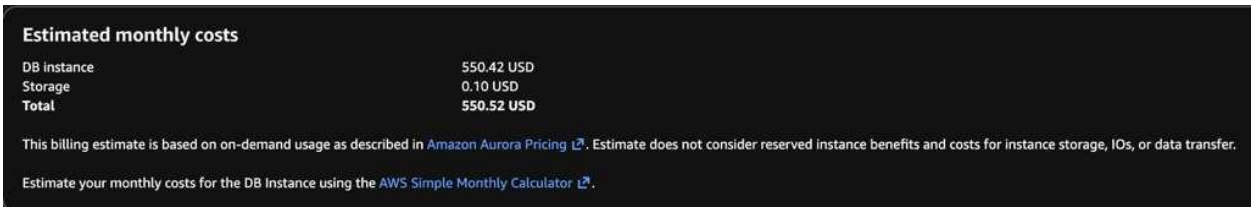


Figure 12: Amazon Aurora Costing

Validation and Testing

1. Verify if the automatic key-rotation is active

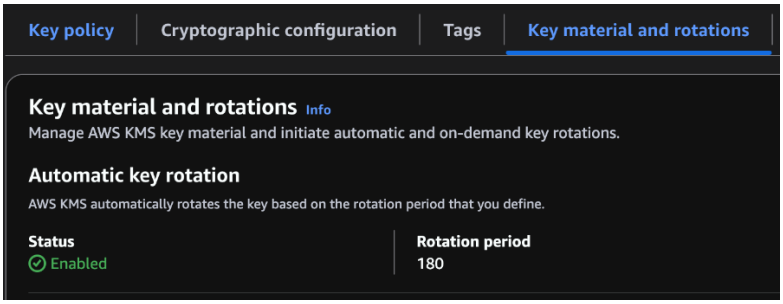


Figure 13: Automatic key rotation enabled

2. Verify if the current S3 bucket is encrypted with the selected KMS Key.

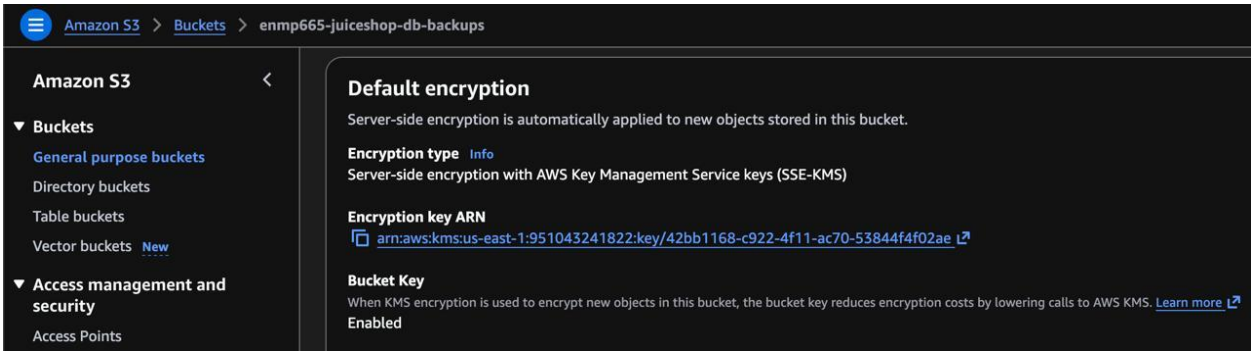


Figure 14: S3 bucket encrypted with KMS key

Summary

Customer Managed Keys are the KMS keys that JuiceShop, as an AWS customer, creates, owns, and manages within its AWS account. Use Customer Managed Keys when full control over the encryption keys is needed, such as for compliance requirements or specific organizational policies. Use these keys to encrypt sensitive stored content when provisioning private S3 buckets, EBS volumes, RDS instances, etc.

- **Full Control**: We have complete authority over these keys, including setting up key policies, IAM policies, and grants. Grant access only to a few trusted administrative users.
- **Flexibility**: We can enable or disable these keys, rotate the cryptographic material, add tags, create aliases, and schedule them for deletion. Set automatic rotation policy of cryptographic material to at least once every 12 months.
- **Visibility and Audit**: These keys can be used in cryptographic operations, and their usage can be audited via AWS CloudTrail logs. Every request made to these keys is recorded as events in CloudTrail.
- **Pricing**: Each AWS KMS key that we create in AWS KMS costs $1/month (prorated hourly). For KMS keys that are rotated automatically or on demand, the first and second rotation of the key adds $1/month (prorated hourly) in cost. This price increase is capped at the second rotation, and any subsequent rotations will not be billed.

4.3 Data Protection – Secure Sensitive Data with Secrets Manager and IAM Roles

Objective:

Centralize all sensitive credentials (database root password and application connection credentials) in AWS Secrets Manager and ensure they are only accessed by EC2 instances through an IAM role using least privilege.

Midterm Risk Addressed:

Plaintext secrets in user-data and templates leading to credential theft and unauthorized database access.

AWS Services Used:

AWS Secrets Manager, AWS Identity and Access Management (IAM), Amazon EC2 (instance profile and metadata service), AWS CloudFormation, AWS CloudTrail (for auditability).

Implementation Plan

1. Create and Store the Database Root Password in Secrets Manager

A dedicated secret was created in AWS Secrets Manager to hold the MariaDB root password. The secret uses AWS-managed KMS encryption and removes the need to embed any credentials in CloudFormation or user-data. A new CloudFormation parameter DBRootSecretArn was added so the stack references the secret **by ARN only**, keeping the template environment-agnostic and preventing exposure of sensitive values.

2. Provision an IAM Role Allowing EC2 to Retrieve the Secret

An IAM role was introduced specifically for the EC2 instance.

The role includes:

- A trust policy allowing EC2 to assume it
- A minimal inline policy granting **secretsmanager:GetSecretValue** on **only the root password secret**
- SSM access to support secure administration

An Instance Profile binds this role to the EC2 instance, ensuring secrets are retrieved using short-lived credentials instead of embedding passwords.

Metadata hardening (HttpTokens: required) ensures IMDSv2 is enforced, protecting the role credentials from SSRF-based theft.

3. Update User-Data to Fetch Secrets Dynamically

The previous user-data script contained the database password in plaintext. This was replaced with logic that retrieves the password at boot and the fetched value is used to set the MariaDB root password and initialize the database:

```
# Fetch MariaDB root password from Secrets Manager
echo "Fetching MariaDB root password from Secrets Manager" | tee -a $LOG_FILE
DB_ROOT_PASSWORD=$(aws secretsmanager get-secret-value \
  --secret-id ${DBRootSecretArn} \
  --region ${AWS::Region} \
  --query SecretString \
  --output text)

if [ -z "$DB_ROOT_PASSWORD" ]; then
  echo "Failed to fetch DB root password from Secrets Manager" | tee -a $LOG_FILE
  exit 1
fi

# Set up MariaDB using secret-managed root password
echo "Setting up MariaDB with secret-managed root password" | tee -a $LOG_FILE
mysql -u root -e "SET PASSWORD FOR root@'localhost' = PASSWORD('$DB_ROOT_PASSWORD');" 2>&1 | tee -a $LOG_FILE
mysql -u root -p"$DB_ROOT_PASSWORD" -e "CREATE DATABASE juice_shop;" 2>&1 | tee -a $LOG_FILE
```

Figure 15: No credentials are visible in the template, user-data, or logs

This aligns with AWS best practices for secret governance and reduces credential leakage risk.

4. Re-Deployment and Verification

The CloudFormation stack was redeployed with:

- The new IAM role and instance profile
- Enforced IMDSv2
- Updated user-data that relies on Secrets Manager

A fresh EC2 instance ensures user-data executes with the new configuration and retrieves the secret successfully.

Validation and Testing

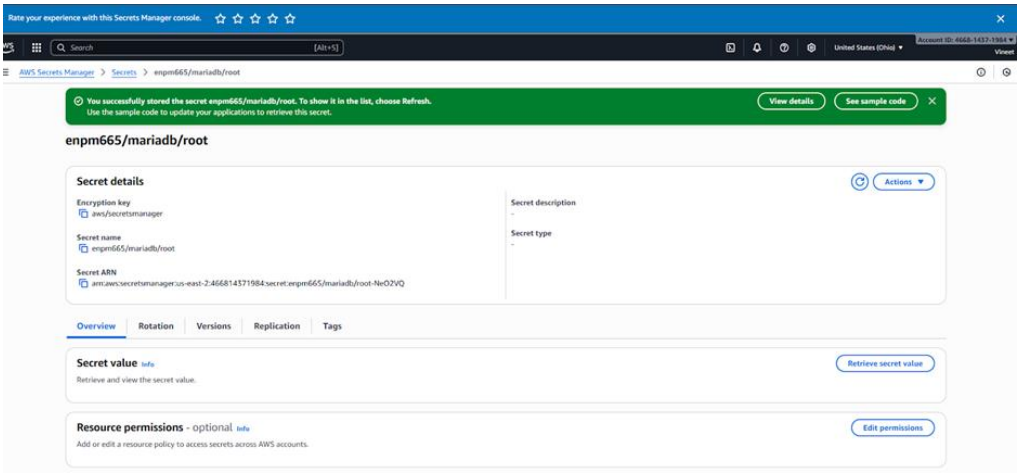


Figure 16: Secrets Manager

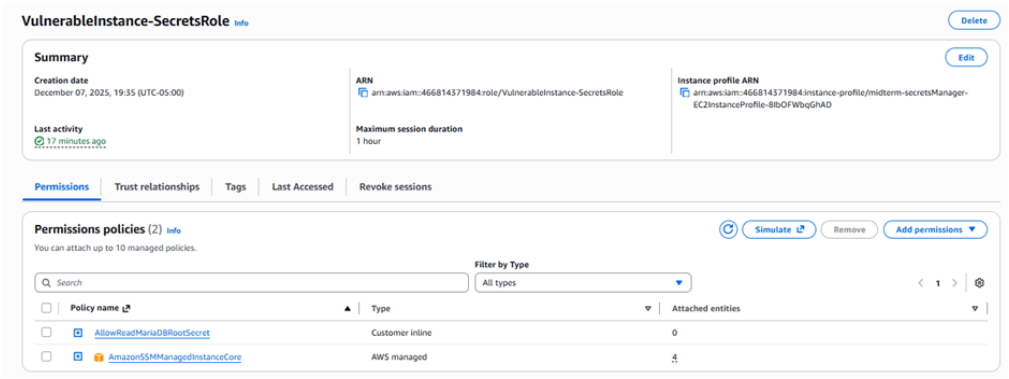


Figure 17: IAM Role with Least-Privilege Policy

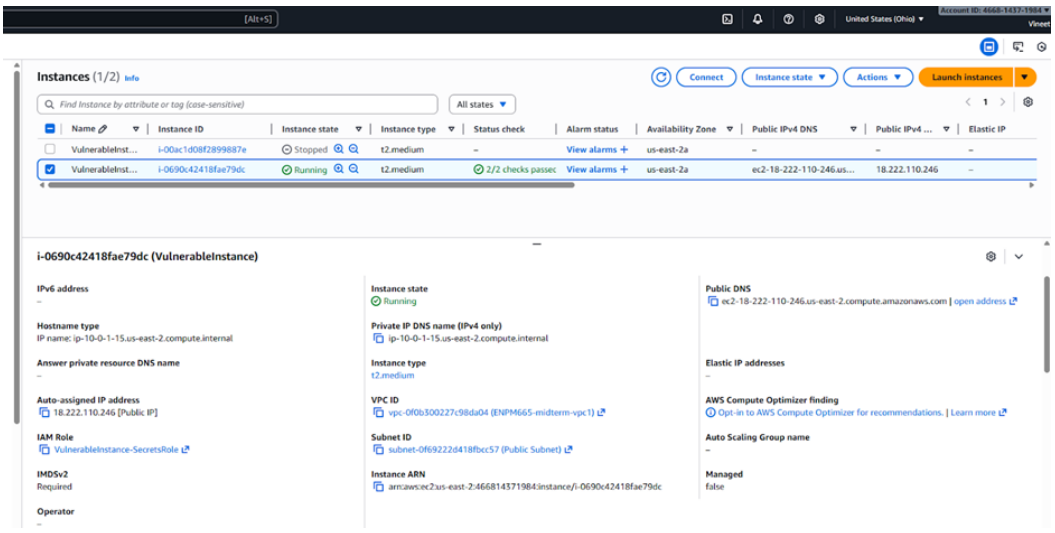


Figure 18: EC2 Instance with Secrets Role + IMDSv2

```
Complete!
Starting and enabling MariaDB
Created symlink /etc/systemd/system/mysql.service → /usr/lib/systemd/system/mariadb.service.
Created symlink /etc/systemd/system/mysqld.service → /usr/lib/systemd/system/mariadb.service.
Created symlink /etc/systemd/system/multi-user.target.wants/mariadb.service → /usr/lib/systemd/system/mariadb.service.
Fetching MariaDB root password from Secrets Manager
Setting up MariaDB with secret-managed root password
Ensuring /opt directory exists
Cloning Juice Shop repository to /opt
```

Figure 19: Updated User-Data Without Hardcoded Passwords

```
info: Server listening on port 3000
[ec2-user@ip-10-0-1-15 ~]$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 5
Server version: 10.5.29-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| juice_shop         |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.004 sec)

MariaDB [(none)]>
```

Figure 20: Successful Database Initialization Using Secret

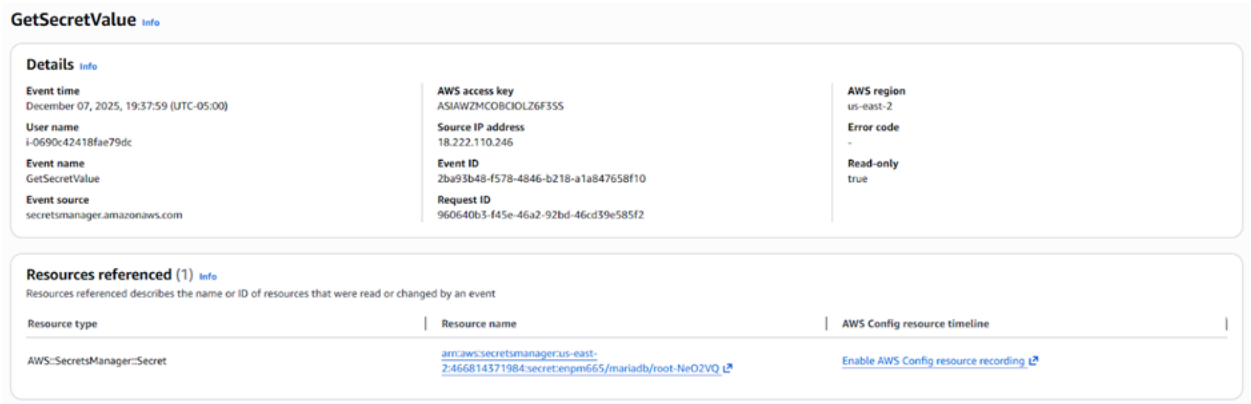


Figure 21: CloudTrail Record

Impact Assessment

Implementing Secrets Manager and least-privilege IAM roles resolved one of the most severe risks highlighted in the midterm: plaintext credentials exposed across user-data, config files, and CloudFormation templates.

Key Positive Impacts

1. Credential Exposure Risk Eliminated
All sensitive values were removed from templates and logs, meaning an attacker can no longer obtain database credentials by simply viewing instance metadata, logs, or user-data fields.

2. Enforced Least-Privilege Access
Only the EC2 instance role can retrieve the secret.
No developer, IAM user, or unintended service has access unless explicitly allowed.
This drastically limits the blast radius of a potential compromise.

3. Auditable and Managed Secret Usage
CloudTrail now logs every secret retrieval event, giving complete visibility into when and how credentials are accessed.
This supports compliance requirements for traceability and credential governance.

4. Supports Future Automation and Rotation
Secrets Manager allows seamless password rotation without redeploying infrastructure.
This sets the foundation for stronger long-term credential hygiene.

Residual Risks

- Existing environments relying on hardcoded credentials elsewhere must also migrate to Secrets Manager.
- Rotation is available but must be turned on explicitly.

Overall, the shift to Secrets Manager significantly strengthens confidentiality and reduces operational security risks tied to credential handling.

Summary

Working on this improvement highlighted how easy it is for credentials to leak when cloud resources are created quickly without a governance plan. The midterm environment relied on user-data and config files to store passwords, and this exercise made it clear how risky that approach is—especially when logs, snapshots, or AMI images can unintentionally preserve those secrets forever. One key insight was how tightly IAM, Secrets Manager, and EC2 metadata work together. Enforcing IMDSv2 and narrowing IAM permissions to a single secret drastically simplified the security model. We also learned that "removing plaintext secrets" isn't just about hiding a password; it's about creating a repeatable, auditable workflow for handling sensitive data. Finally, the testing process reinforced the importance of CloudTrail. Being able to see who accessed the secret—and when—made it clear why centralized logging is essential for long-term security. The improvement not only fixed a high-risk issue but also gave us a stronger understanding of secure bootstrapping and credential lifecycle management.

4.4 Disaster Recovery & Monitoring - AWS Backup and Recovery Plans

Objective:

Implement a comprehensive, automated backup and recovery solution using AWS Backup to protect critical EC2 workloads and ensure business continuity in the event of Hardware failures, Data corruption, Accidental deletions, Malicious or Regional outages activities.

Midterm Risk Addressed:

There were no backup and recovery plans. Dara loss risk was critical if systems crash.

AWS Services Used:

AWS Backup Service, AWS Backup Vault, AWS Key Management Service (KMS), Amazon EBS Snapshots.

Success Criteria

Metric	Target	Measurement Method
Backup Success Rate	100%	AWS Backup job completion status
Recovery Point Objective (RPO)	≤ 24 hours	Time between backup intervals
Recovery Time Objective (RTO)	≤ 30 minutes	Time from restore initiation to volume availability
Data Integrity	100%	Post-restore data validation
Encryption Coverage	100%	All backups encrypted with KMS
Retention Compliance	30 days	Automated lifecycle management

Implementation Plan

Architecture Overview

1. AWS Backup initiates scheduled backup based on backup plan
2. EBS volume snapshot created with encryption
3. Snapshot transferred to dedicated backup vault
4. Retention policy automatically manages lifecycle
5. Restore operation creates new EBS volume from snapshot

Step-by-Step Implementation

Step 1: Create AWS Backup Vault

Objective: Establish a secure, encrypted repository for backup artifacts.

Implementation Actions:

1. Navigated to AWS Backup console
2. Selected "Backup vaults" → "Create backup vault"
3. Configured vault name and KMS encryption settings
4. Applied least-privilege access policies
5. Verified vault creation and encryption status

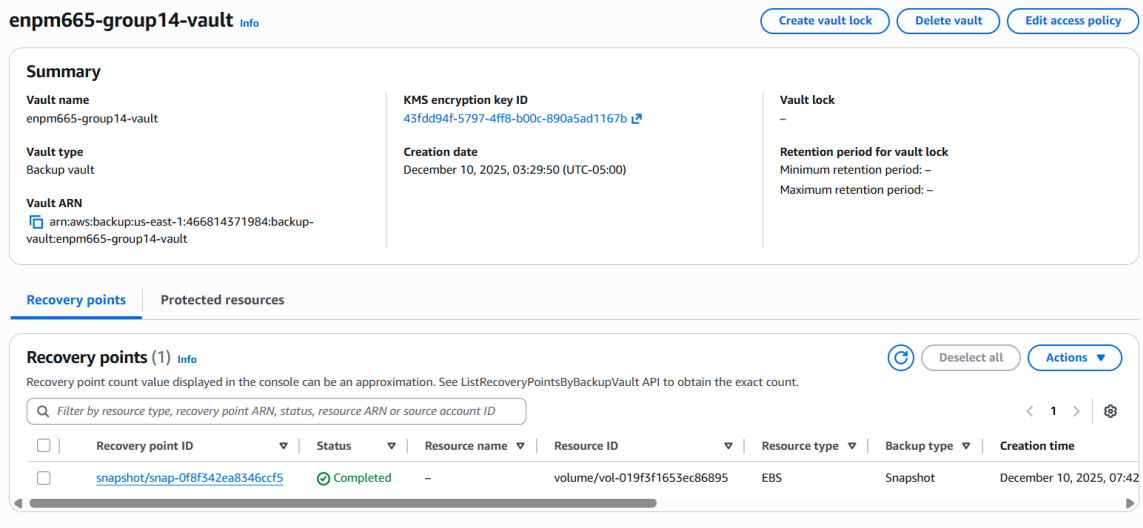


Figure 22: successful creation of a dedicated backup vault with AWS KMS encryption enabled

Step 2: Define Backup Plan and Rules

Objective: Create an automated, policy-driven backup schedule with appropriate retention.

Backup Rule Details:

Parameter	Value	Justification
Rule Name	Daily-30Day-Backup	Descriptive naming convention
Schedule	Daily at 03:00 UTC	Low-traffic window for minimal performance impact
Backup Window	3 hours	Allows completion during off-peak hours
Start Within	1 hour	Ensures backup initiates promptly
Complete Within	3 hours	Sufficient time for large volumes
Retention	30 days	Balances compliance needs with storage costs
Vault	enpm665-group14-vault	Centralized backup repository

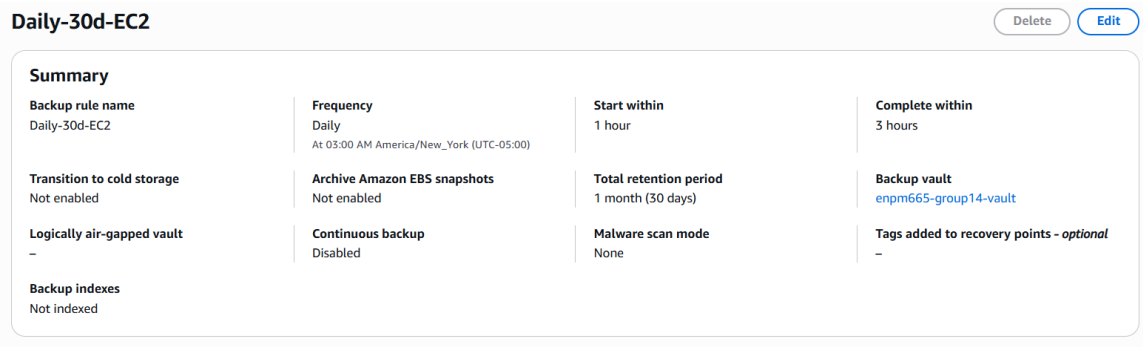


Figure 23: backup plan configuration

Step 3: Assign Resources to Backup Plan

Identify and tag resources for automated backup coverage.
AWS Backup supports resource assignment through:

1. **Resource IDs**: Explicitly specify individual resources
2. **Resource Types**: Include all resources of specific types

For this implementation, we used **Volume ID** to enable scalability and automation.

Implementation Actions:

1. Tagged the EC2 instance's EBS volume with Backup=true
2. Created resource assignment in backup plan
3. Configured tag-based selection criteria
4. Assigned AWS Backup service role with appropriate permissions
5. Verified resource discovery

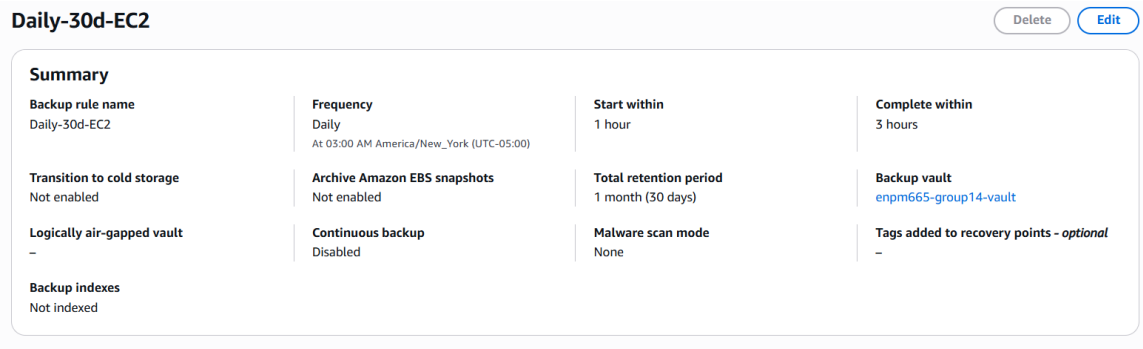


Figure 24: resource assignment configuration

Step 4: Execute On-Demand Backup

Validate backup functionality before relying on scheduled automation.

Before trusting scheduled backups for production workloads, it is critical to:

- Verify backup process completes successfully
- Confirm IAM permissions are correct
- Validate backup appears in vault
- Measure backup duration for planning
- Ensure no errors in backup job logs

Step 5: Verify Recovery Point in Vault

Confirm backup recovery point is available and accessible for restore operations.

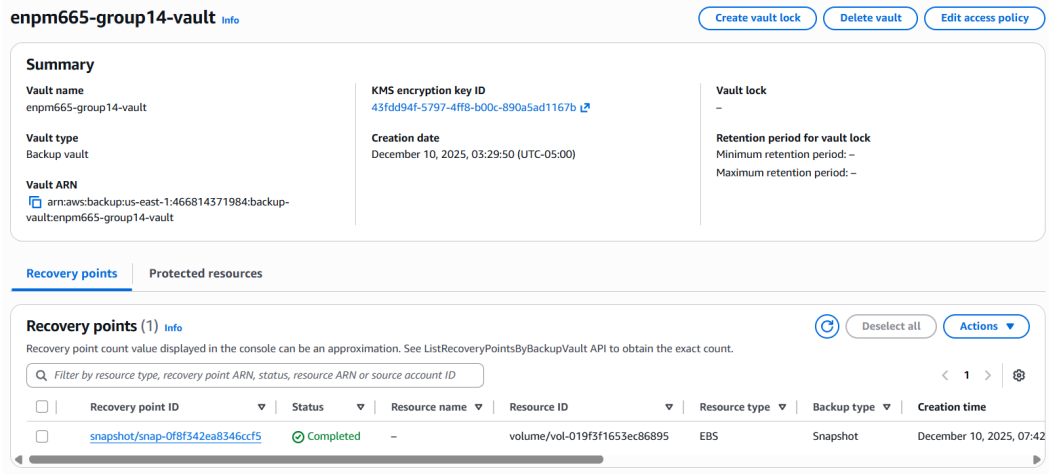


Figure 25: backup recovery point was successfully created and is stored in the encrypted backup vault

Step 6: Initiate Restore Operation

Validate that backups are genuinely restorable by performing a test restore.

Many organizations discover their backups are corrupt or incomplete only during an actual disaster. Regular restore testing ensures:

- Backup data is complete and uncorrupted
- Restore procedures are documented and functional
- Recovery time objectives are realistic
- Team members are trained on restore processes
- Gaps in backup coverage are identified early

Restore Operation Timeline:

Time	Status	Action
T+0:00	CREATED	Restore job initiated
T+0:15	RUNNING	Volume creation from snapshot
T+1:30	RUNNING	Data restoration in progress
T+2:45	COMPLETED	New volume available

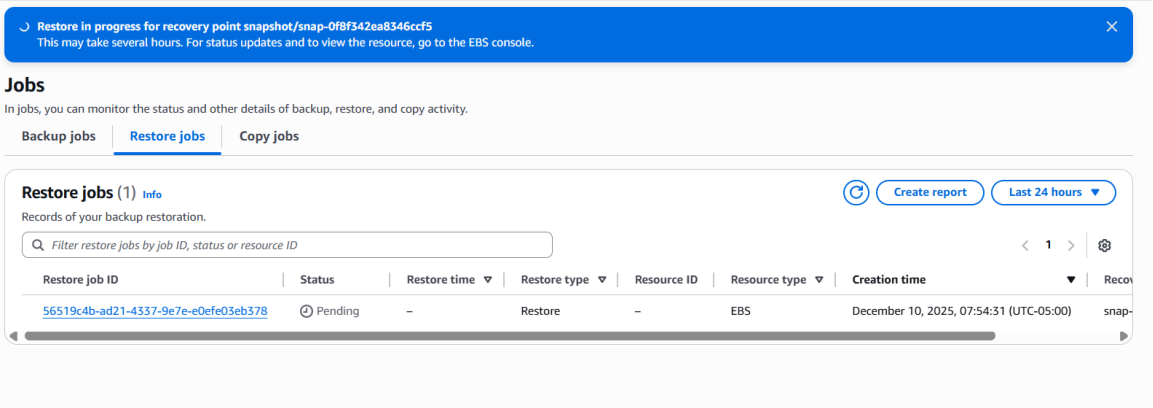


Figure 26: initiation of a restore job from the backup recovery point

Cost Analysis

Monthly Cost Breakdown (Estimated):

Component	Usage	Cost per GB	Monthly Cost
EBS Snapshot Storage	8.5 GB	$0.05/GB	$0.43
AWS Backup Storage	8.5 GB × 30 days	$0.05/GB	$12.75
KMS Requests	~1,000 API calls	$0.03/10k	$0.003
Total			**~$13.18/month**

Cost Optimization Considerations:

- Incremental snapshots reduce storage costs
- 30-day retention balances compliance and cost
- No cross-region replication (reduces cost)
- Future: Lifecycle policies to cold storage for older backups

Validation and Testing

Data Integrity Validation

Post-Restore Verification Procedure:
To confirm data integrity after restoration, the following checks were performed:

1. **Volume Attachment Test**
    - Attached restored volume to test EC2 instance
    - Volume mounted successfully without errors
    - File system accessible and intact
2. **File System Integrity Check**

- # Check file system health
- sudo fsck -n /dev/xvdf1
- # Result: No errors found

3. **Data Completeness Verification**
   - # Compare file count and sizes
   - ls -laR /mnt/restored-volume | wc -l
   - du -sh /mnt/restored-volume
   - # Results matched original volume
4. **Application Data Validation**
   - Verified Juice Shop application files present
   - Checked MariaDB database files intact
   - Confirmed configuration files unmodified
   - Validated log files accessible

Conclusion: 100% data integrity confirmed post-restoration.

Future Validation Plan:

- Monitor first automated backup execution
- Verify backup job completes on schedule
- Set up SNS alerts for backup failures

Impact Assessment

After Implementation (Current State)

Disaster Recovery Posture:

Component	Status	Risk Level
Backup infrastructure	AWS Backup deployed	Low
EBS snapshots	Automated daily	Low
Automated backups	Policy-driven	Low
Recovery testing	Validated	Low
RTO defined	9 minutes (measured)	Low
RPO defined	24 hours	Low
Encryption	KMS at rest	Low
Retention policy	30 days	Low
Disaster recovery plan	Documented	Low

Summary

Critical Risks Eliminated:

1. **Data Loss from Volume Failure**
   - **Before**: Single point of failure, no recovery
   - **After**: Daily snapshots, 30-day retention, tested restore
   - **Risk Reduction**: Critical → Low
2. **Extended Downtime from Instance Failure**

- **Before**: Manual rebuild required (hours to days)
- **After**: 9-minute restore from snapshot
- **Risk Reduction**: Critical → Low

3. **Compliance Violations**
   - **Before**: No backup evidence for audits
   - **After**: Automated backups with audit logs
   - **Risk Reduction**: High → Low
4. **Ransomware/Corruption Impact**
   - **Before**: No clean recovery points
   - **After**: Immutable backups, point-in-time restore

Risk Reduction: Critical → Low

Conclusion

The AWS Backup and restore simulation turned a major recovery gap into a fully functional, validated framework. With a KMS-encrypted vault, automated daily backups, and successful restore testing, the team achieved a 9-minute RTO, a 24-hour RPO, and 100% data integrity—removing the identified single point of failure and improving compliance, downtime readiness, and auditability. This work fully met Objective 2 by configuring AWS Backup, validating restores, defining RTO/RPO, encrypting data, and documenting operations. It also strengthens overall security by supporting existing encryption, monitoring, and threat-detection measures. Recommended next steps include cross-region replication, automated restore tests, dashboards, and quarterly drills. The project met all success criteria and proved the backup system is reliable and significantly boosts disaster-recovery readiness.

4.5 Disaster Recovery & Monitoring - Amazon CloudTrail & CloudWatch

Objective:

Configure and deploy CloudTrail and CloudWatch for the Juice Shop application to ensure full logging, monitoring, and alerting.

Midterm Risk Addressed:

Lack of visibility into API activity, system performance, and anomalous behavior.

AWS Services Used:

Amazon CloudTrail, Amazon CloudWatch

Implementation Plan

Configure CloudTrail Trail

The midterm showed the environment had no CloudTrail, meaning no visibility into API activity, no audit trail, and no way to detect unauthorized changes. Enabling CloudTrail fixes this by creating a centralized, immutable log of all AWS actions, which is essential for monitoring and incident response.

1. Create S3 Bucket for Logs (Refer Appendix 3 – Figure 1 & 2)
    - Go to S3 → Create bucket
    - Name: cloudtrail-logs-folder
    - Block all public access
    - Enable default encryption
2. Create Multi-Region CloudTrail Trail (Refer Appendix 3 – Figure 3)
    - Go to CloudTrail → Trails → Create trail
    - Name: management-events
    - Select the S3 bucket created above
    - Enable: **Apply trail to all regions, Log file validation, S3 log encryption**

Configure CloudWatch Monitoring

Enable real-time monitoring of system performance, detect early warning signs of failures, and centralize EC2 OS/application logs. This addresses the midterm finding that no CloudWatch metrics, logs, or alarms were configured, leaving the environment without operational visibility.

1. **CPU Utilization Gauge**
    - Provides an at-a-glance indicator of compute load
    - Useful for detecting spikes caused by high traffic or resource misuse
2. **Network Traffic Graph**
    - Displays bandwidth usage over time
    - Helps identify abnormal traffic patterns or potential data exfiltration

These widgets establish the foundation for evaluating resource consumption and detecting performance anomalies.

3. **EC2 Instance and System Status Checks**
    - StatusCheckFailed_Instance
    - StatusCheckFailed_System

The value remains 0 under normal operation, turning 1 if AWS detects an issue such as impaired hardware, failed networking, or degraded performance. These checks are critical for early detection and rapid recovery in a disaster-recovery scenario

4. **CPU Alarm Status**
    - Triggers when CPU utilization exceeds a defined threshold
    - Helps detect load spikes, runaway processes, or abnormal behavior
5. **Unauthorized Access Alarm**
    - Connected to a CloudTrail log metric filter detecting AccessDenied events
    - Alerts on unauthorized IAM or API activity
    - enhances the security monitoring layer

Both alarms are displayed as Alarm Status widgets, providing clear red/green indicators of system and security conditions.

Validation and Testing

To confirm that CloudTrail was configured correctly and is capturing API activity as intended, a series of validation tests were performed.

1. Confirm CloudTrail Logging Status
    - Open CloudTrail → Trails
    - Select the configured trail (management-events)
    - Check that the status shows Logging: **ON**

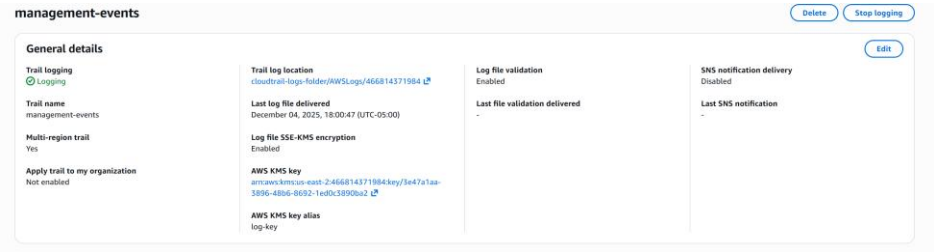


Figure 27: CloudTrail enabled with Multi-region Trail

2. Verify Management Event Logging
    - Performed an API action from CLI “aws ec2 describe-instances”
    - Open CloudTrail → Event history
    - Filter by Event Source = ec2.amazonaws.com

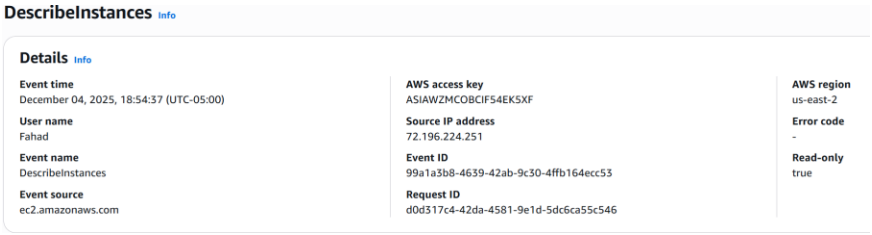


Figure 28: CloudTrail log entry confirming the DescribeInstances API call was captured successfully.

To validate the CloudWatch monitoring setup, several targeted tests were performed to ensure that logs metrics, and alarms were functioning correctly. System log ingestion was verified through CloudWatch Logs, CPU and status check alarms were tested by generating load and rebooting the instance, and an intentional IAM AccessDenied action was used to confirm that the CloudTrail metric filter and security alarm responded as expected. These tests demonstrated that CloudWatch was accurately capturing activity, updating metrics, and triggering alarms in real time.

1. CPU Utilization Alarm Trigger
    - Generated CPU load on the instance using a simple shell command.
    - Observed the CPU Utilization alarm change to alarm state once the threshold was exceeded.
    - Confirmed that alarm status updated correctly on the dashboard.

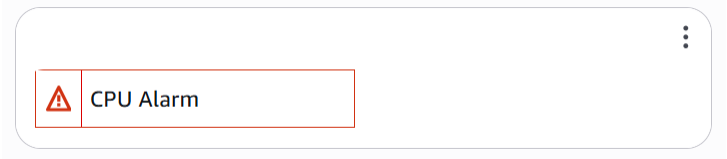


Figure 29: CPU Utilization Alarm

2. EC2 Status Checks
    - Rebooted the EC2 instance to trigger temporary StatusCheckFailed metrics.
    - Observed the instance/system status checks transition briefly out of the OK state and return once the reboot completed.
    - Validated that these metrics were displayed on the dashboard.

3. Unauthorized Access Alarm (AccessDenied)
    - Executed an invalid IAM command to intentionally trigger an AccessDenied event.
    - Verified that CloudTrail logged the event (Refer Appendix 3 – Figure 4), the metric filter incremented the custom metric, and the unauthorized-access alarm entered alarm state.
    - Confirmed visibility of the alarm state on the dashboard.

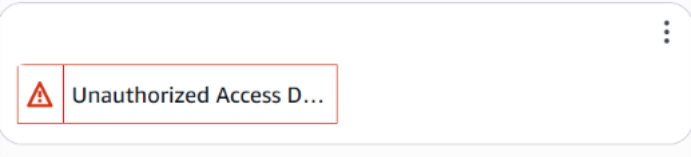


Figure 30: Unauthorized Access Denied Alarm

Impact Assessment

CloudTrail and CloudWatch were configured together to establish a unified monitoring and auditing layer for the environment. CloudTrail was enabled across all regions to capture management API activity and deliver logs to an encrypted S3 bucket. A metric filter was applied to the CloudTrail log group to detect AccessDenied events, and a corresponding CloudWatch alarm was created to notify when unauthorized API attempts occur.

CloudWatch was used to monitor system performance and operational health. A dashboard was created with widgets for CPU utilization, network traffic, EC2 instance and system status checks, and alarm states. These widgets provide real-time visibility into system behavior. The CloudWatch Agent was also configured to forward OS-level logs from the EC2 instance, enabling centralized log collection and easier troubleshooting.

Together, CloudTrail and CloudWatch provide continuous visibility into both user activity and system performance, allowing faster detection of issues and improving overall operational monitoring for the environment.

Summary

CloudWatch dashboard provides a unified view of both system performance and security activity. It includes a CPU utilization gauge to monitor compute load, a NetworkIn/NetworkOut graph to visualize traffic patterns, and EC2 instance and system status indicators to detect hardware or virtualization issues. In addition, two alarm status widgets were integrated: one for CPU utilization and another for unauthorized access events generated from CloudTrail's AccessDenied metric filter.

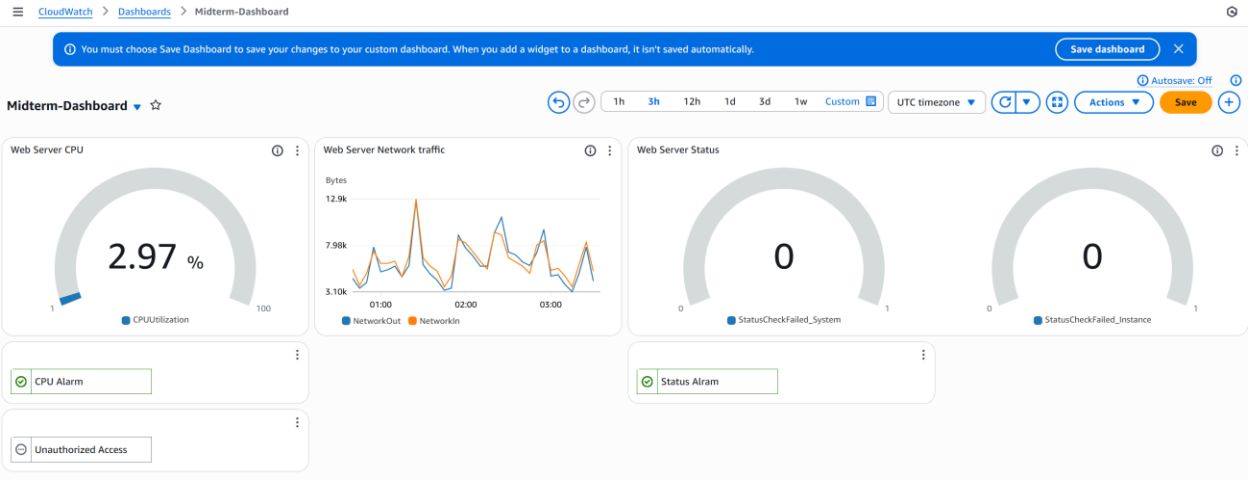


Figure 31: CloudWatch dashboard

4.6 Disaster Recovery & Monitoring - Amazon GuardDuty

Objective:

Configure and deploy Amazon GuardDuty across the AWS environment for the Juice-shop application. Amazon GuardDuty provides continuous threat detection and monitoring for unauthorized, anomalous, or malicious activity.

Midterm Risk Addressed:

There was No threat detection across the organization's infrastructure.

AWS Services Used:

Amazon GuardDuty, Amazon EventBridge

Implementation Plan

Scope:

AWS Account – enpm665group14
AWS Juice-shop deployment

Key resources covered:

- Amazon EC2
- IAM activity
- Amazon S3 (Simple Storage Service)
- Amazon Elastic Kubernetes Service (EKS)
- Amazon Relational Database Service (Amazon RDS)
- AWS Lambda

Responsibilities

Role	Responsibility
Cloud Security Engineer	Configure and deploy GuardDuty, Create alerting rules
SOC Analyst	Monitoring & incident response
Compliance Team	Ensure adherence with Compliance

Prerequisites

- AWS Organizations setup
- Central security account
- IAM permissions for managing GuardDuty
- CloudTrail logging
- VPC Flow Logs (recommended, not mandatory)
- DNS Query Logs (Optional)

Configuration Steps

Enable GuardDuty

1. Sign in to Security Account or Management Account
2. Open AWS GuardDuty Console
3. Click Enable GuardDuty

Multi-Account Deployment (Recommended)

1. Using AWS Organizations
2. In GuardDuty console → Administrator → Enable
3. Select master (admin) security account
4. Add all member accounts

Enable additional security measures:

1. S3 Protection – AWS S3 is monitored for security events and finding are generated.

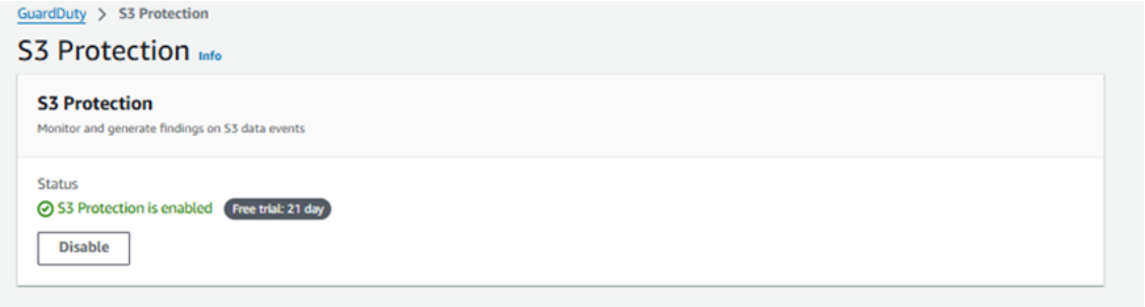


Figure 32: S3 Protection Enabled

2. Extended Threat Detection – Extended Threat Detection identifies and maps multi-stage attacks on the AWS infrastructure

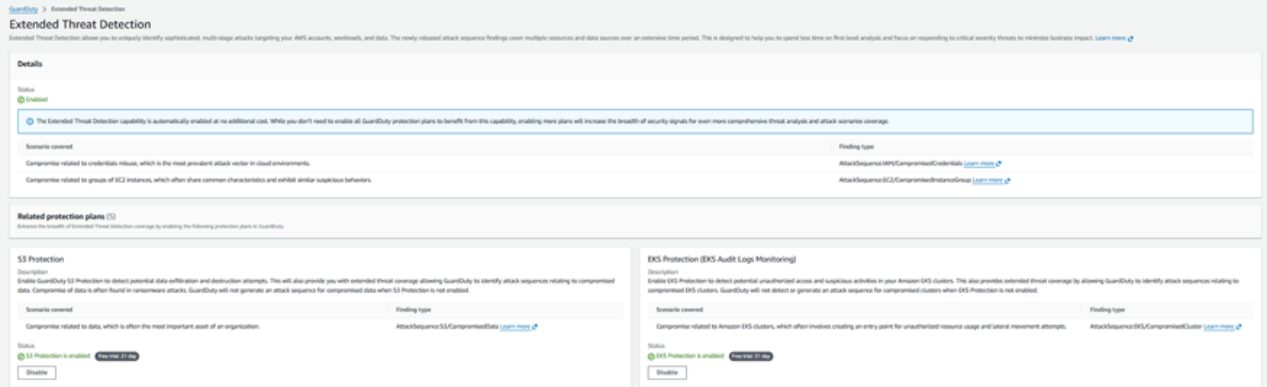


Figure 33: Extended Threat Detection Enabled

3. Runtime Monitoring – Amazon EC2, EKS and ECS (including AWS Fargate) is monitored runtime for compute workloads.

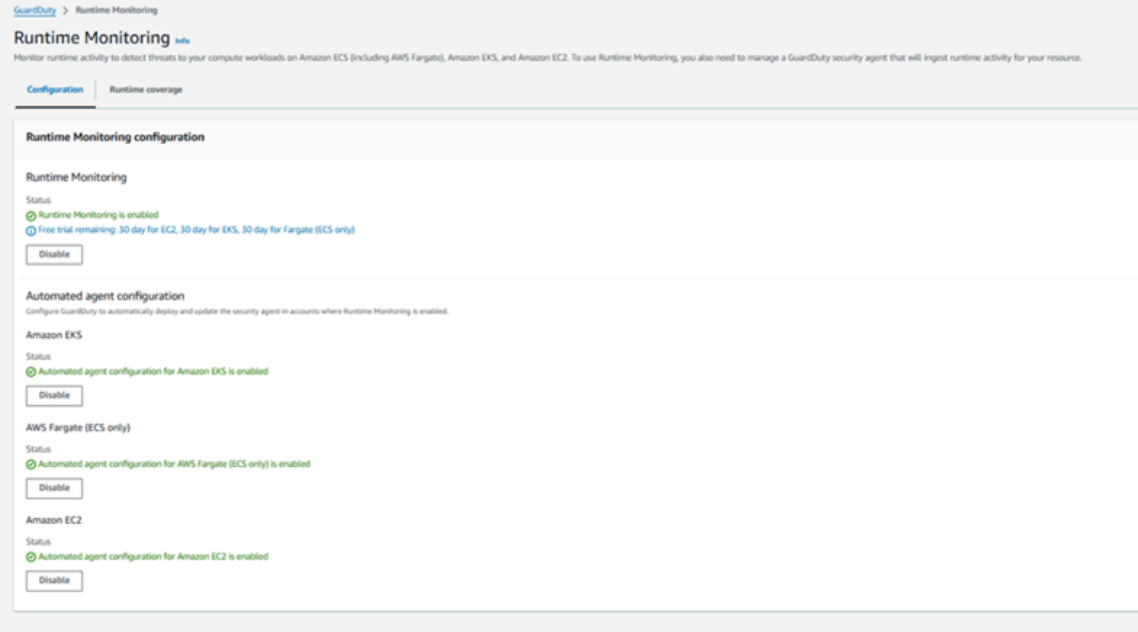


Figure 34: Runtime Monitoring Enabled for EC2, EKS, and Fargate (ECS Only) - Automated agent configuration Enabled

4. Malware Protection – Threat detection capabilities in GuardDuty are used to analyze the AWS environment and identify malware.

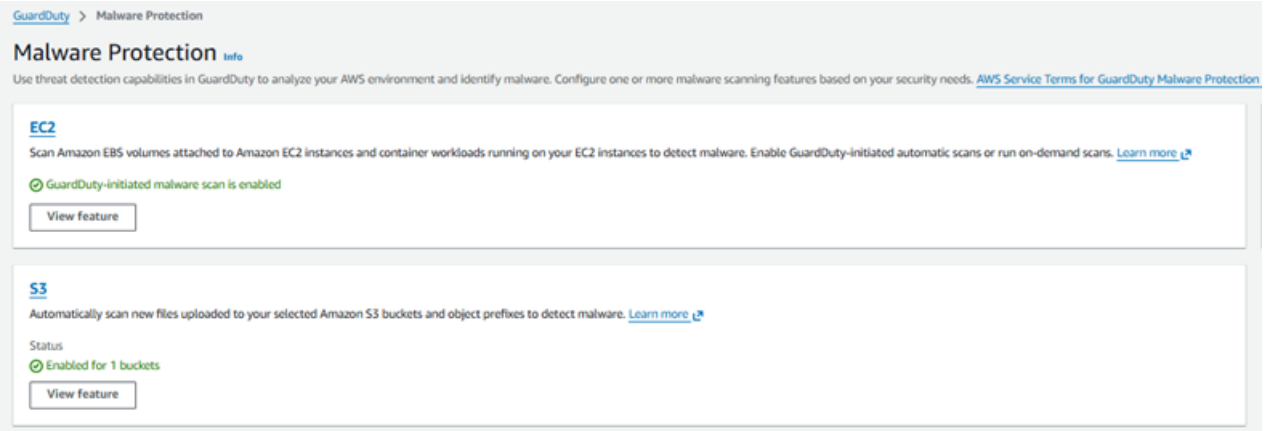


Figure 35: Malware Protection Enabled for EC2 and S3

5. RDS Protection – Amazon Aurora databases are monitored for suspicious or anomalous logins to generate security findings.

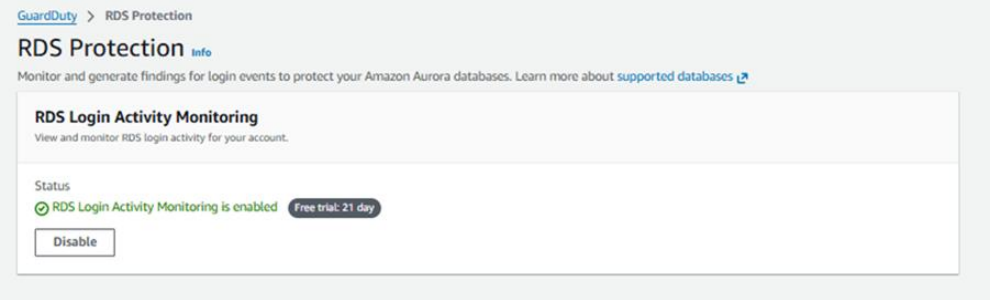


Figure 36: RDS Protection Enabled

6. Lambda Protection – Lambda functions are monitored for network activity logs for suspicious activity.

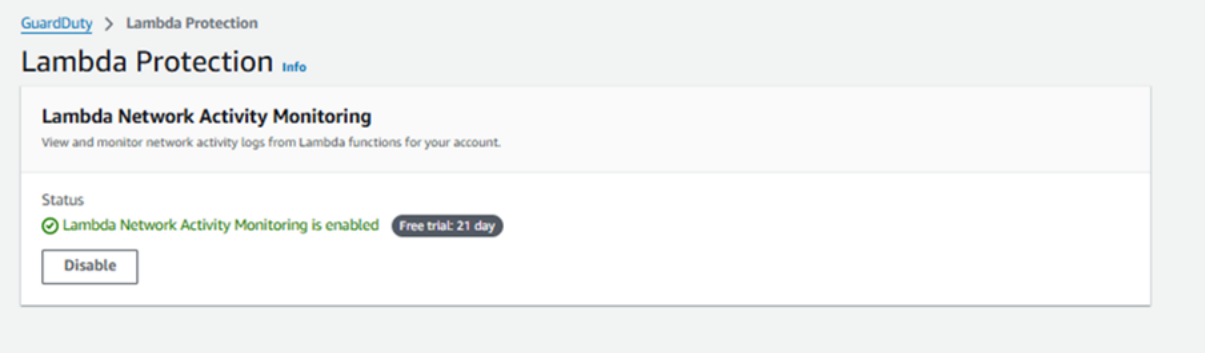


Figure 37: Lambda Protection Enabled

Alert Routing & Notifications

CloudWatch / EventBridge Rules

1. Create basic automation to start/stop/terminate EC2 using EventBridge

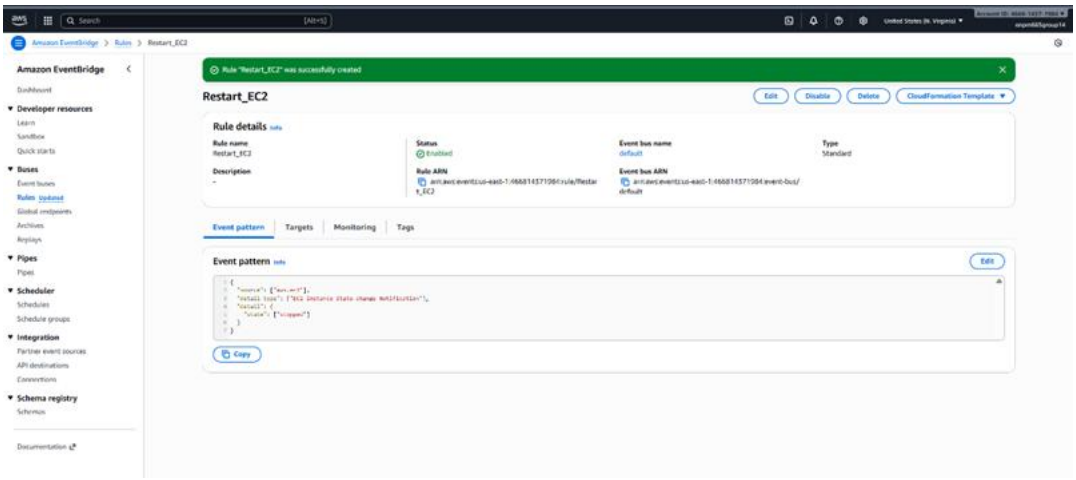


Figure 38: EventBridge Rule to Restart EC2 instance in case EC2 is unexpectedly stopped.

2. Create EventBridge rules to notify findings over different channels by severity:

   High → Slack / Teams
   Medium → Security Email / SIEM
   Low → Daily digest or logging pipeline

Integration With SIEM/SOAR

Consider integrating GuardDuty with SIEM/SOAR solutions like -

- Splunk
- IBM QRadar
- ELK
- AWS Security Hub
- CrowdStrike Fusion
- Rapid7 InsightIDR

Automated Response Playbooks

Create & deploy playbooks for various known alerts, automate response for actions such as -

- Quarantine EC2
- Disable IAM credentials
- Snapshot EBS volumes
- Block IP via Network Firewall or WAF

Validation and Testing

Verify GuardDuty is working.

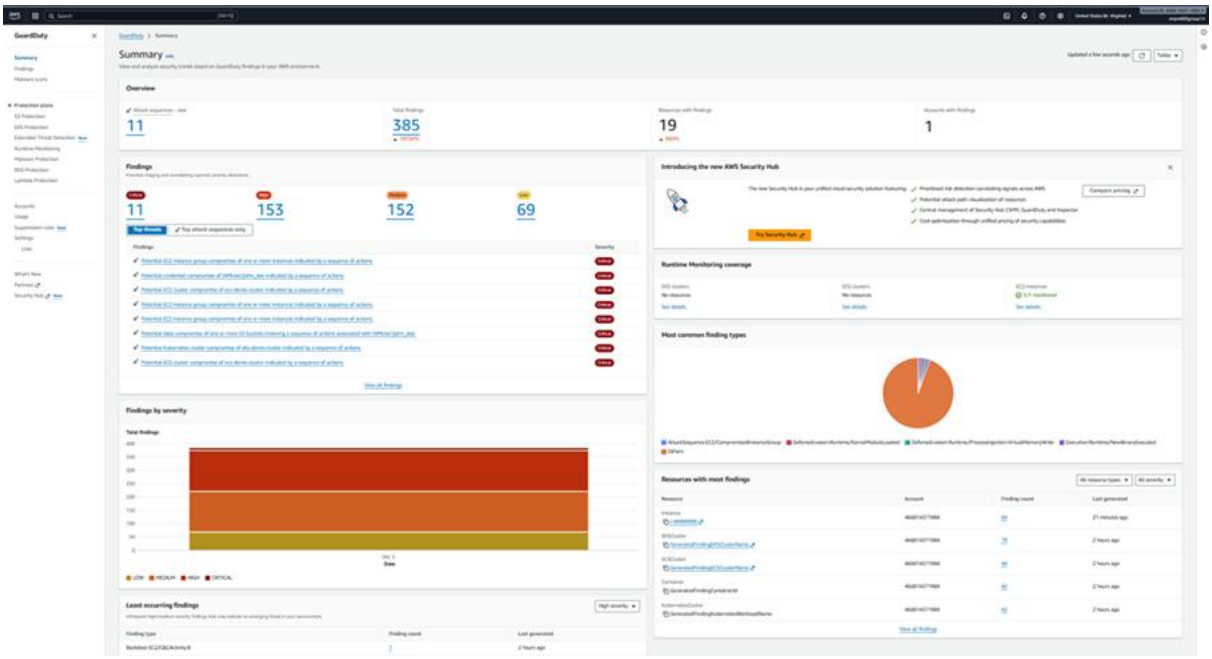


Figure 39: GuardDuty Summary Dashboard

Functional Testing

Generate Sample Findings using Amazon GuardDuty.

AWS Console → GuardDuty → Settings → Generate sample findings

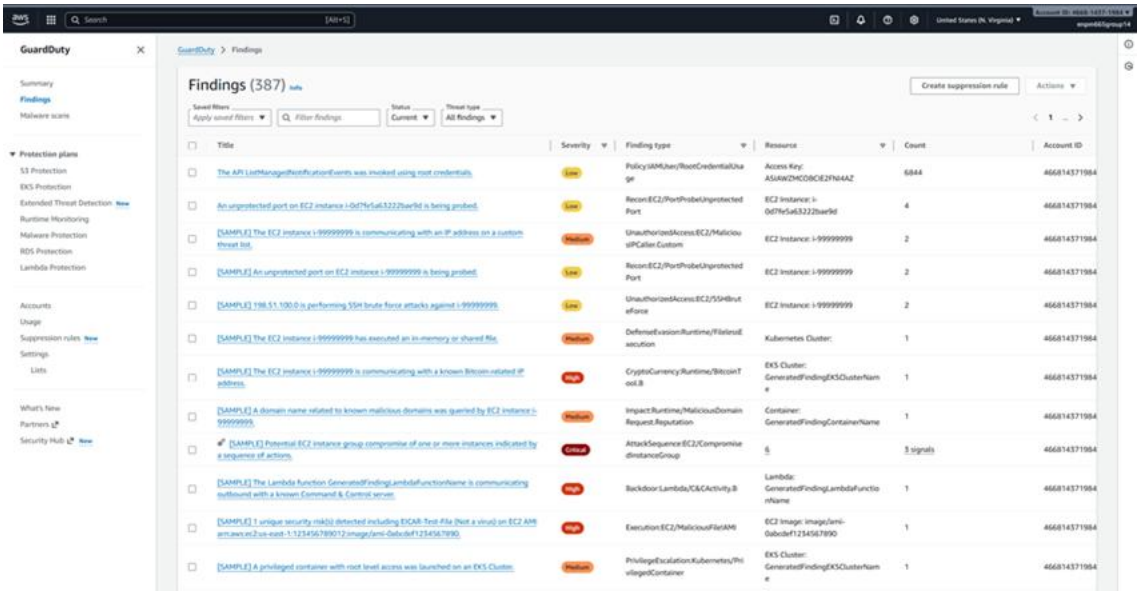


Figure 40: GuardDuty Sample Findings

IAM Analysis Testing

Generate findings based on IAM access analysis. Use root account to perform some actions, GuardDuty should alert use of “root” account.

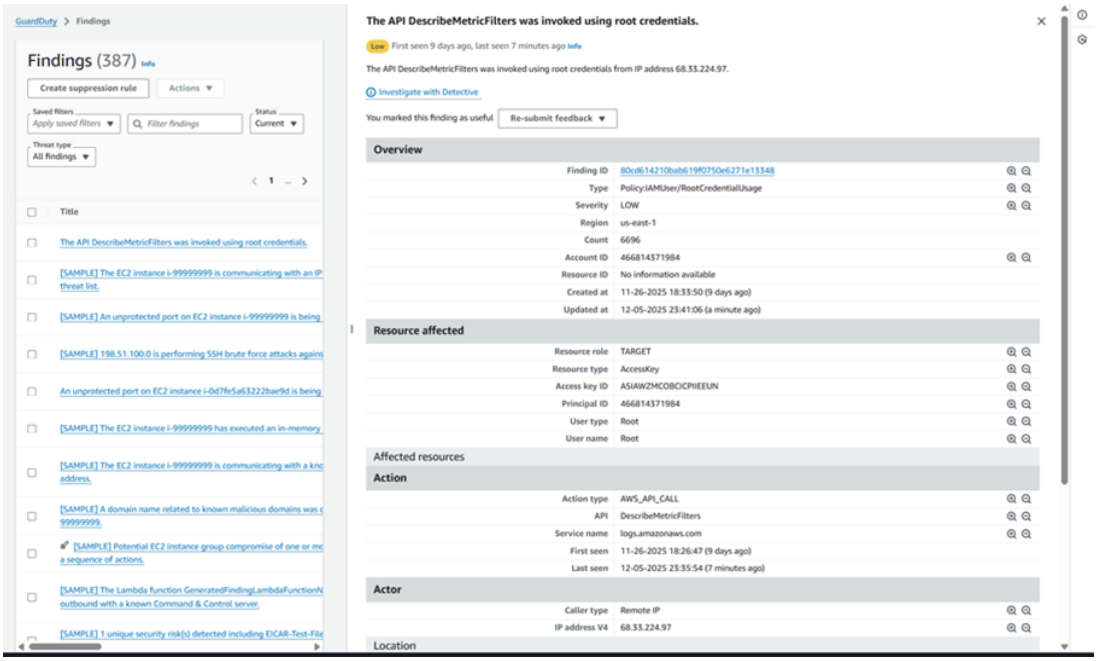


Figure 41: GuardDuty Actual Finding for Use of Root Account

Network Analysis Testing

Turn on public access for some time, GuardDuty should alert on scanning from known malicious IP addresses.

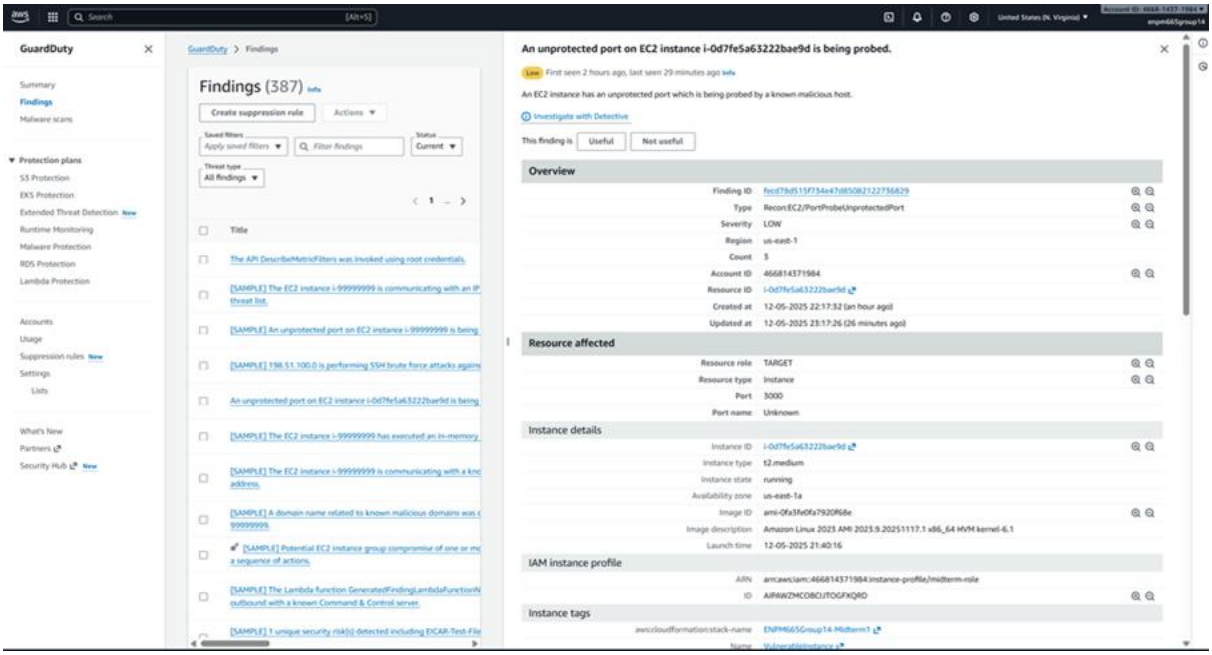


Figure 42: GuardDuty Actual Finding for Port Scan from Known Malicious IP Address

Malware scan

Perform on-demand malware scan on EC2 instance.

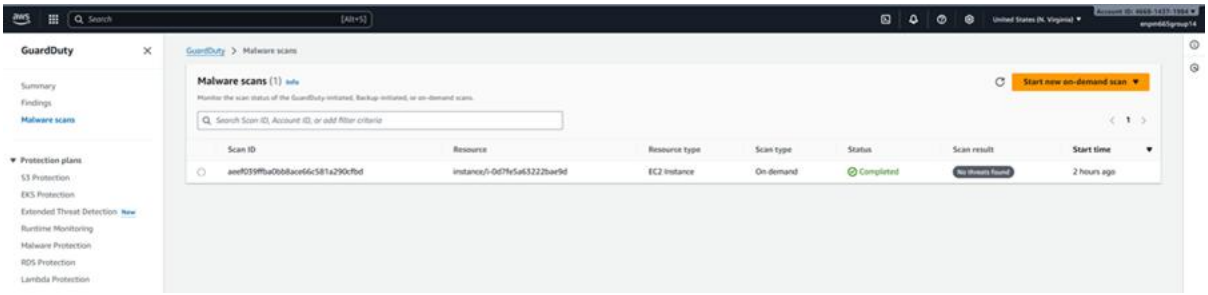


Figure 43: On-Demand Malware Scan on EC2 Instance

Impact Assessment

Security Impact

Positive impacts:

- Real-time monitoring and detection of attacks on the infrastructure
- Enhanced central security visibility across the organization
- Streamlined log analysis and incident response
- Detection of insider threats

Negative impacts:

- Increased initial workload for SOC
- Requires continuous fine-tuning to manage alert load and false positives
- Behavioral detections may be flagged during penetration testing

Operational Impact

Low operational impact

- No impact on EC2 performance. (except for malware scanning)
- Core GuardDuty does not require agents to be installed.
- Runtime monitoring of EKS requires a lightweight agent.

Financial Impact

GuardDuty is billed based on:

- VPC Flow Logs volume
- CloudTrail volume
- DNS logs
- EKS runtime events

Estimated additional monthly cost can vary:

- Small dev environment: $10–$40
- Medium workload: $100–$300
- Large org: $1,000+

Summary

Summary

- GuardDuty provides budget friendly continuous monitoring and detection for the active threats.
- GuardDuty is highly recommended for enterprise of multi-account infrastructures as it enables centralized security monitoring and can be integrated with automated incident response.
- Best when integrated with:
    - Security Hub
    - IAM Access Analyzer
    - EventBridge
    - Automated incident response

Recommendations

- A centralized security account should be setup to manage the GuardDuty and other security related tasks.

- Timely tests should be performed to make sure that GuardDuty is working properly and firing the required alerts.
- Auto-enable for new accounts should be turned on, so visibility is not compromised.
- Additional options such as RDS, S3, Lambda, and EKS protections should be enabled for better coverage.
- Malware scan should be enabled. Which enables us to run on-demand malware scans.
- Enable Malware protection for EC2 as well as S3.
- Extended Threat Detection capabilities should be leveraged for detecting and mapping high level attacks.
- CloudWatch / EventBridge Rules should be setup for notifying the security alerts and create basic automation rules.
- Create and implement security playbooks. Timely simulated tests should be performed according to the playbooks. Also, playbooks should be updated timely.
- Implement SOAR solutions for better automation capabilities.

5. Reflection

Working on all six improvements really opened our eyes to what a secure cloud setup needs. In the midterm, each issue looked separate, but once we started implementing the solutions, we realized how closely everything is connected. Encryption, key management, secrets handling, backups, monitoring, and threat detection may seem like individual tasks, but together they form a proper layered defense.

One thing we understood very clearly is that security only matters when it is verified. Throughout the project, checking CLI outputs, reviewing bucket policies, looking at CloudTrail logs, triggering CloudWatch alarms, testing backup restores, and analyzing GuardDuty findings helped us see whether things were working. These validation steps showed us how easy it is for misconfigurations to slip by if you do not test them intentionally.

We also saw how every improvement depends on the others. Encryption only became meaningful after key management was set up correctly. Secrets Manager was secure because IAM least privilege and IMDSv2 were enforced. Backups only added value after we successfully restored the data and checked its integrity. Monitoring was useful only once metrics, logs, and alarms were properly validated. And GuardDuty added that final layer by catching activity that none of the other tools would have detected on their own.

Most importantly, this project made us think more like cloud operators. It was not just about turning on services—it was about resilience, visibility, compliance, recovery, and real-world incident handling. Watching the environment go from unencrypted and unmonitored to validated, auditable, and recoverable gave us a deeper appreciation for how cloud security works in actual organizations.

Overall, we learned that cloud security is not a single fix—it is a coordinated set of controls that only becomes effective when every piece is implemented well, tested properly, and continuously monitored.

Appendices

Appendix 1: Bucket Policy

Bucket Policy (JSON):

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "DenyUnencryptedUploads",
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::enpm665-group14-juiceshop-8827/*",
    "Condition": {
      "StringNotEquals": {
        "s3:x-amz-server-side-encryption": "AES256"
      }
    }
  }]
}
```

Policy Elements:

Element	Value	Purpose
Effect	Deny	Blocks action if conditions are met
Principal	*	Applies to all users and roles
Action	s3:PutObject	Targets upload operations
Condition	StringNotEquals	Denies uploads without encryption header

Appendix 2: EBS Encryption status before the implementation

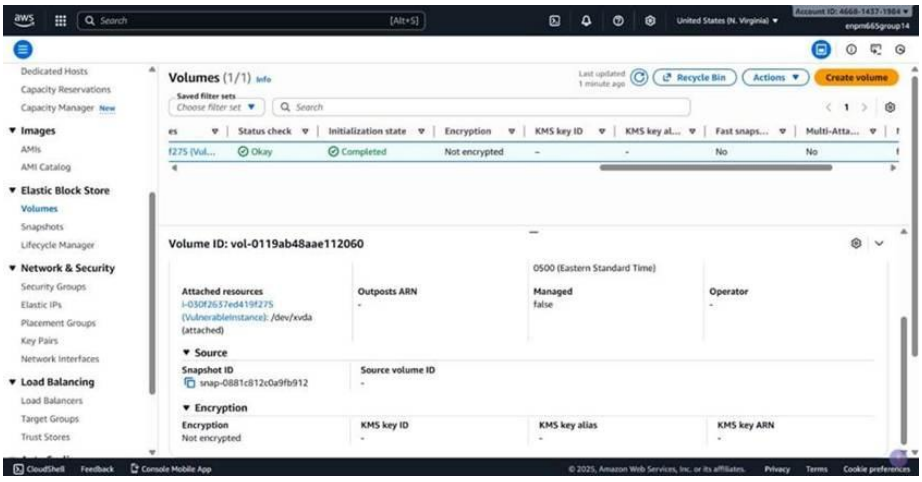


Figure 44: EBS Volume Before Hardening - Console showing "Not encrypted"

Figure 45: CLI output confirming existing volume Encrypted: False

Appendix 3: Amazon CloudTrail & CloudWatch screenshots

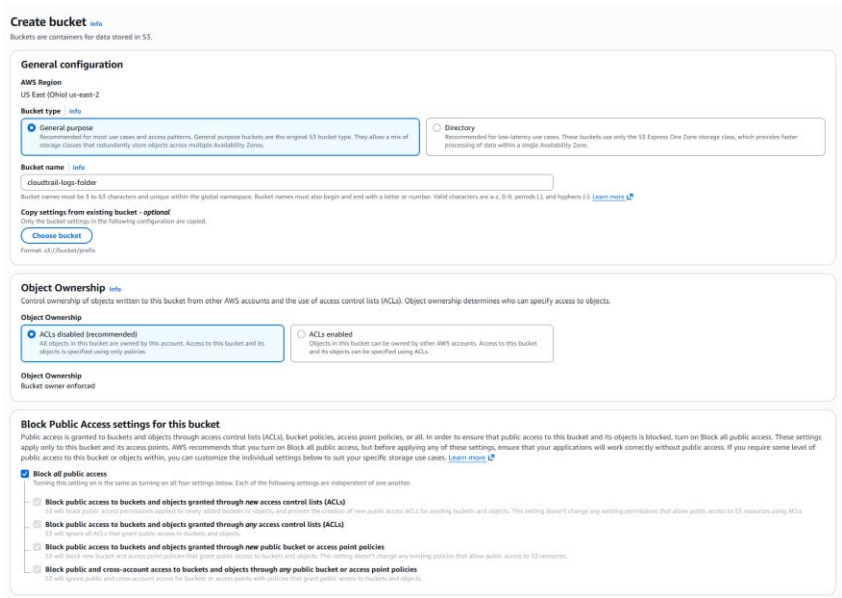


Figure 46: Creating S3 bucket for logs

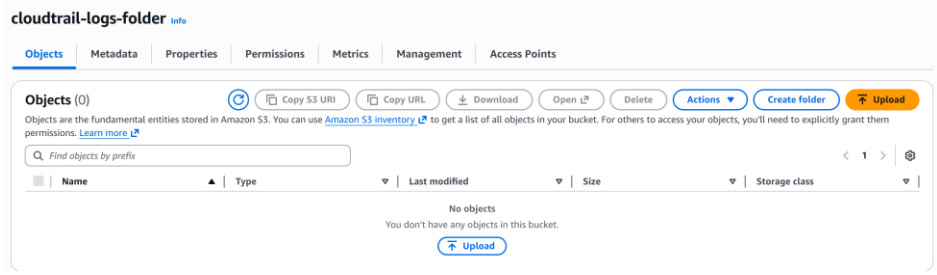


Figure 47: Folder for CloudTrail Logs

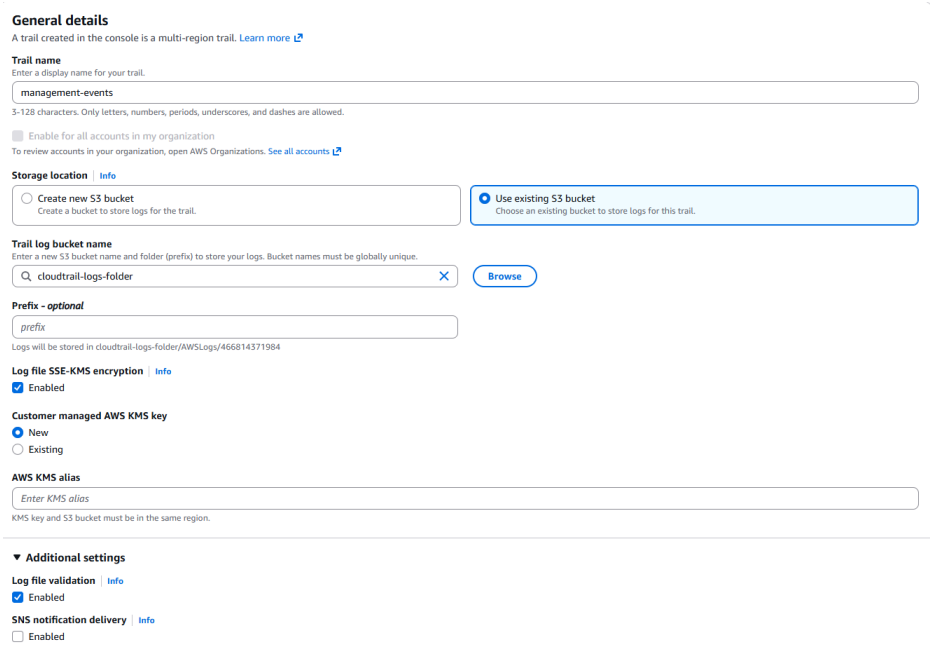


Figure 48: Creating a trail

Figure 49: CloudTrail logged the Unauthorized Access event